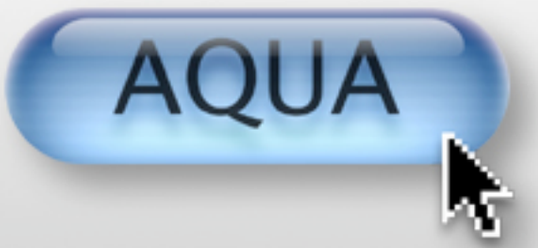
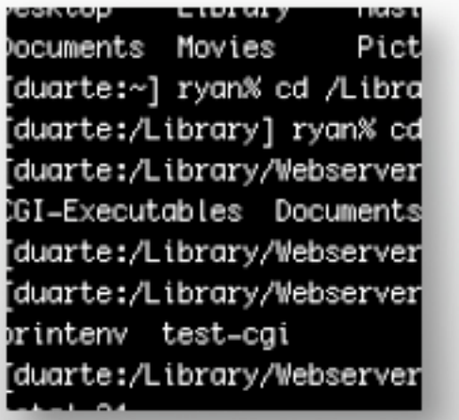
Data Exchange for Carbon Applications

Session 415





Data Exchange for Carbon Applications

Christopher Linn
Senior Software Engineer

Carbon Data Exchange Today

- Scrap Manager
- Drag Manager
- Translation Manager
- OSType data identifiers

Problems

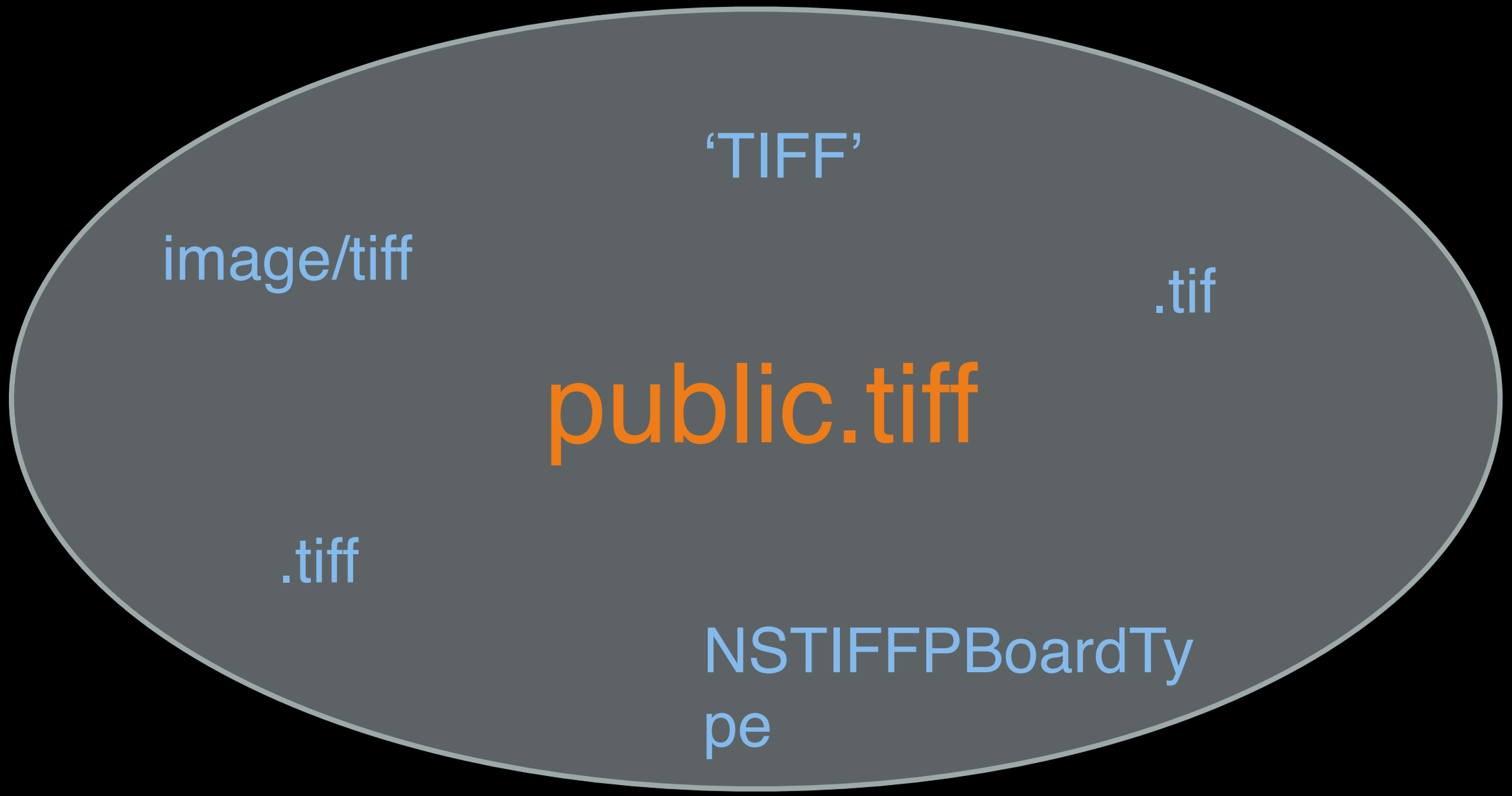
- Separate APIs for Scrap and Drag
  - Double the work for app developers
- Translation
  - Difficult API
  - Not Core Foundation-friendly
- Type identification is a mess
  - Multiple namespaces
  - Extensibility issues

Solutions in Panther

- Carbon Pasteboard
  - One API for moving data between apps
- Translation Services
  - Improved architecture
- Uniform Type Identifiers
  - A foundation for the future

Uniform Type Identifiers

What Is a TIFF?



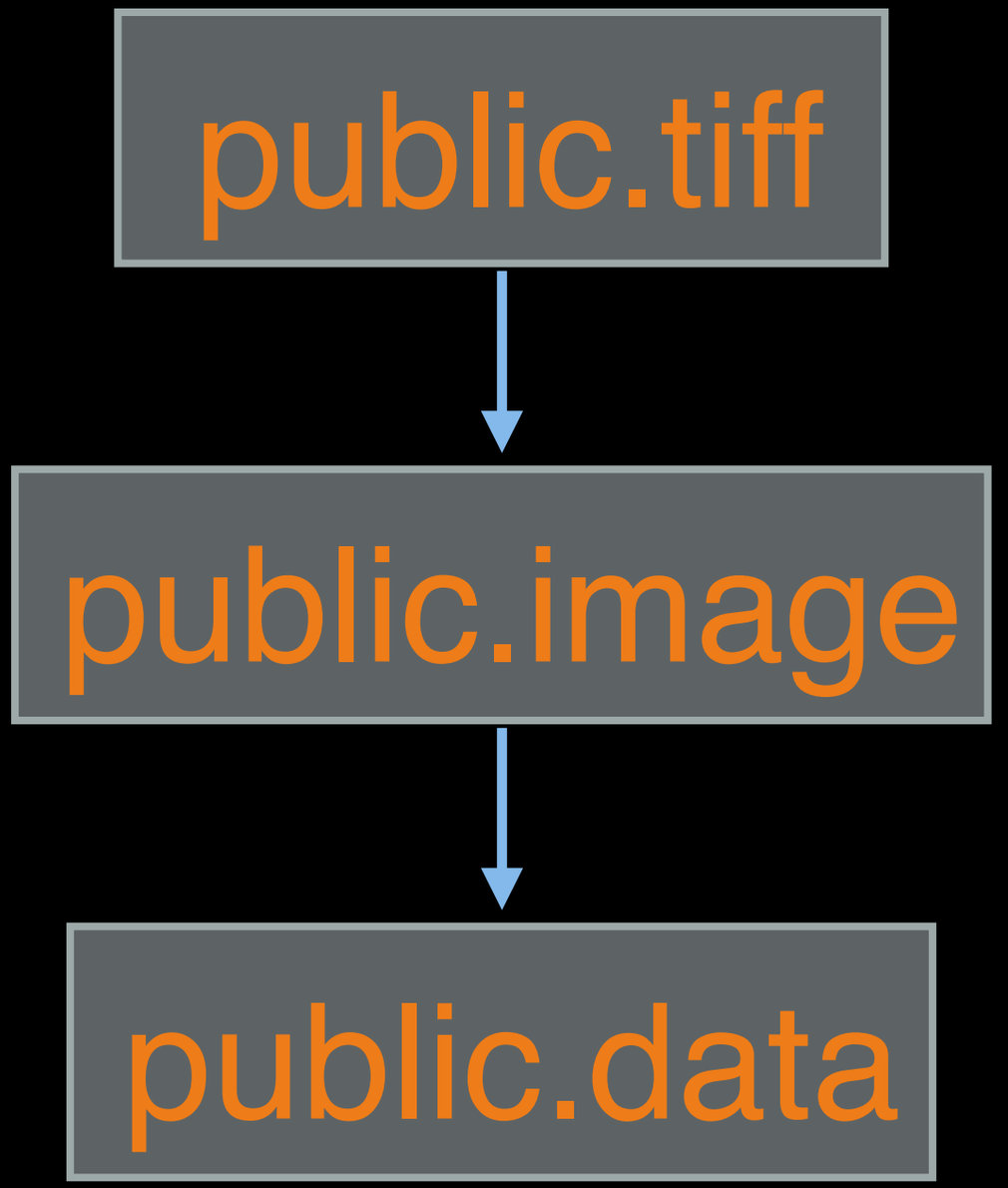
Uniform Type Identifiers

- Reverse-DNS names prevent collisions
- No centralized name registry
- Standard types are placed in the public domain
- Abbreviated as UTI

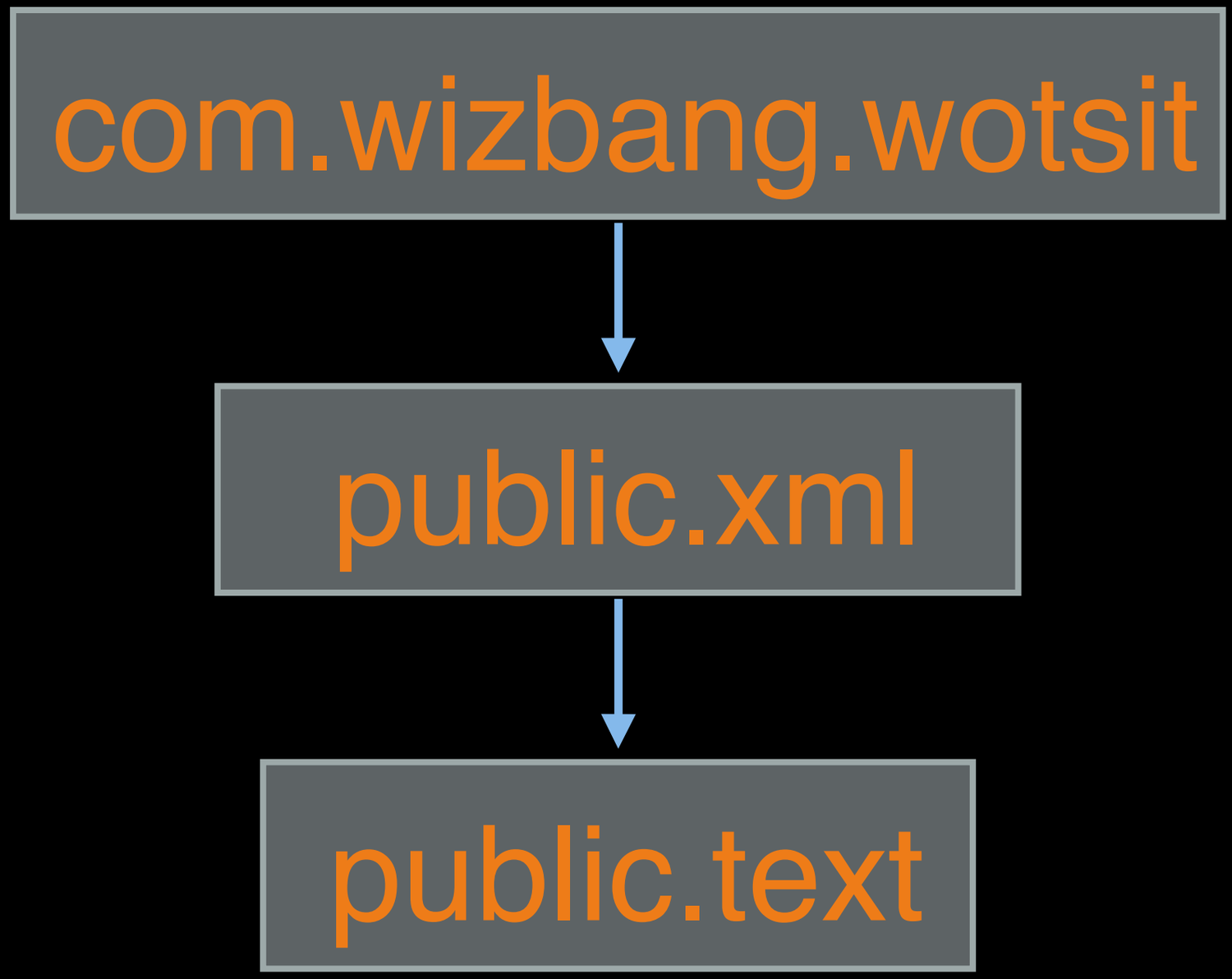
Examples

com.adobe.psd

What Else Is a TIFF?



Why Is Conformance Important?



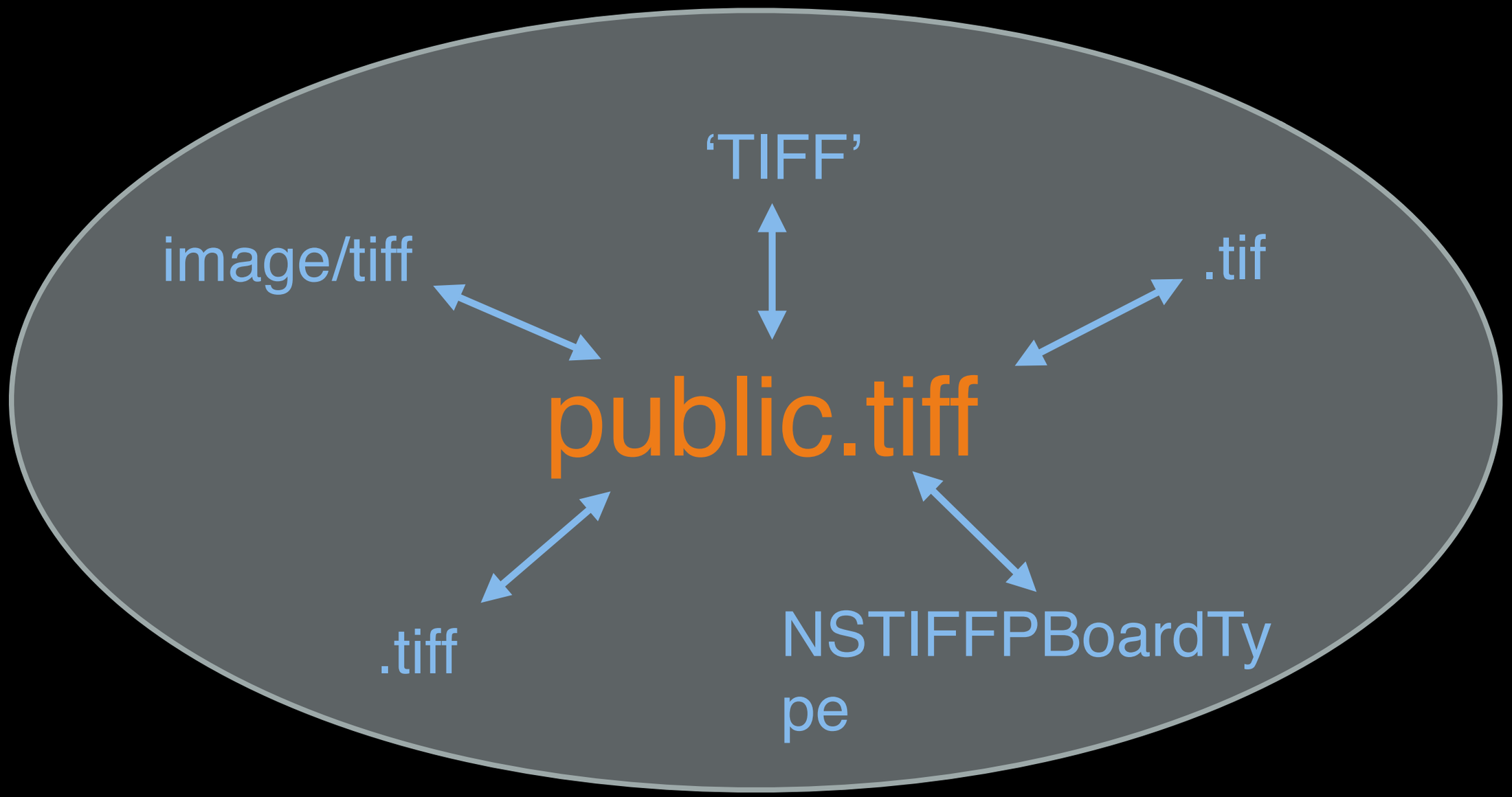
Uniform Type Identifier Goals

- Unify all type identifier spaces
- Model type conformance
- Enable open extensibility
- Provide a lightweight, foundational API

Tags

- Tags are alternate identifiers for a type
- Each tag has a class
  - OSType
  - Filename extension
  - Etc.
- Not necessarily a one-to-one mapping

Tags



Example: OSType to UTI

```
CFStringRef ostype;
CFStringRef uti;
CFStringRef mimetype

ostype = CFSTR( “JPEG” );

uti = UTTypeCreatePreferredIdentifierForTag(
            kUTTagClassOSType, tag, NULL );
```

Example: OSType to UTI…to MIME

```
CFStringRef ostype;
CFStringRef uti;
CFStringRef mimetype

ostype = CFSTR( “JPEG” );

uti = UTTypeCreatePreferredIdentifierForTag(
            kUTTagClassOSType, tag, NULL );

mimetype = UTTypeCopyPreferredTagWithClass(
                uti, kUTTagClassMIMEType );
```

Conformance vs. Equality

Conformance vs. Equality

```
Boolean UTTypeConformsTo(
  CFStringRef   inTypeIdentifier,
  CFStringRef   inConformsToTypeIdentifier );

Boolean UTTypeEqual(
  CFStringRef   inTypeIdentifier1,
  CFStringRef   inTypeIdentifier2 );
```

Declaring New Types

- Types are declared in bundle property lists
- Style is similar to bundle document type declarations
  - See /System/Library/CoreServices/SystemIcons.bundle
- Registered types have global scope
- Exported vs. Imported types

Guidelines

- Use the API to test UTIs for conformance and equality
- Use the built-in public types where possible
- Declare new types in your Internet domain space
- See LaunchServices.h for API details

Carbon Pasteboard

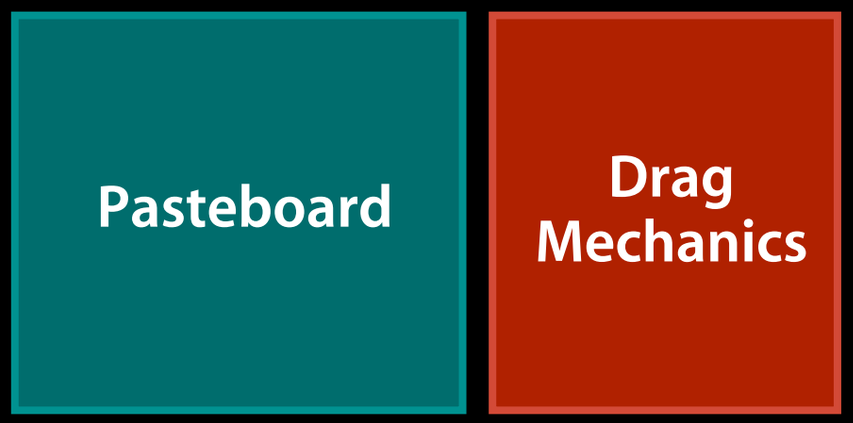
Bryan Prusha
HIToolbox Engineer

Data Exchange Today

- Separate Scrap and Drag Managers
- Different models
- Limited typing mechanism
- Nonexistent communication with Cocoa
- Labor intensive memory model

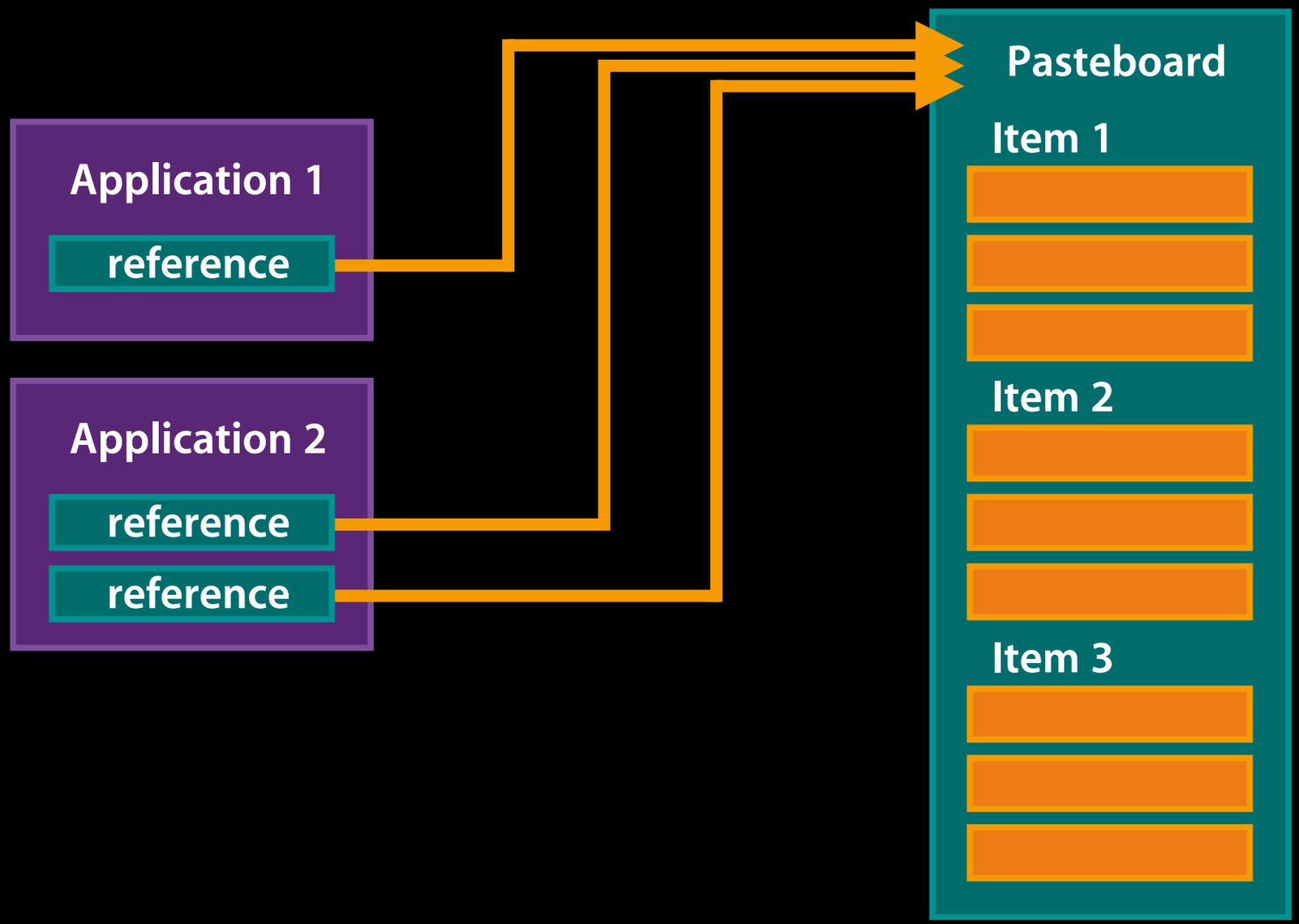
Pasteboard Overview

- Unification of Scrap and Drag Flavor APIs
- Supports copy and paste, drag and drop, and services
- Multiple items, each with set of flavors
- Flavor types are based on Uniform Type Identifiers
  - Compatible with NSPasteboard
- Core Foundation based



Pasteboard Reference

- Proxy to global pasteboard resource



Creating Pasteboards

- Created by name
  - Standard
  - Unique
  - Custom

```
typedef struct OpaquePasteboardRef* PasteboardRef;

#define kPasteboardClipboard CFSTR(“com.apple.pasteboard.clipboard”)
#define kPasteboardFind      CFSTR(“com.apple.pasteboard.find”)
#define kPasteboardUnique    (CFStringRef)NULL

OSStatus PasteboardCreate(
            CFStringRef     inName,   // can be NULL
            PasteboardRef*  outPasteboard );
```

Pasteboard Access

- Sync pasteboard before reading current data
- Clear pasteboard before writing new data

```
PasteboardSyncFlags PasteboardSynchronize(
      PasteboardRef inPasteboard );

OSStatus PasteboardClear(
      PasteboardRef inPasteboard );
```

Receiving Items

- Pasteboards contain multiple items
- Accessed via unique IDs

```
OSStatus PasteboardGetItemCount(
            PasteboardRef     inPasteboard,
            ItemCount*        outItemCount );

typedef void* PasteboardItemID;

OSStatus PasteboardGetItemIdentifier(
            PasteboardRef     inPasteboard,
            UInt32            inIndex,
            PasteboardItemID* outItem );
```

Receiving Flavor Types

- All flavor types for an item are retrieved at once
- Always use UTI comparison routines
- Translations offered after sender flavors

```
OSStatus PasteboardCopyItemFlavors(
            PasteboardRef      inPasteboard,
            PasteboardItemID   inItem,
            CFArrayRef*        outFlavorTypes );
```

Receiving Flavor Flags

- Flags are similar to those from Scrap and Drag
- “Request only” supports private flavors between apps

```
typedef UInt32 PasteboardFlavorFlags;
enum {
    kPasteboardFlavorNoFlags                = 0,
    kPasteboardFlavorSenderOnly             = (1<<0),
    kPasteboardFlavorSenderTranslated       = (1<<1),
    kPasteboardFlavorNotSaved               = (1<<2),
    kPasteboardFlavorRequestOnly            = (1<<3),
    kPasteboardFlavorSystemTranslated       = (1<<8),
    kPasteboardFlavorPromised               = (1<<9)
};

OSStatus PasteboardGetItemFlavorFlags(
            PasteboardRef           inPasteboard,
            PasteboardItemID        inItem,
            CFStringRef             inFlavorType,
            PasteboardFlavorFlags*  outFlags );
```

Receiving Data

- Based on CFData

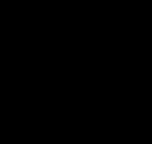
```
OSStatus PasteboardCopyItemFlavorData(
            PasteboardRef    inPasteboard,
            PasteboardItemID inItem,
            CFStringRef      inFlavorType,
            CFDataRef*       outData );
```

Adding Data

- Add data in your application's order of preference
- kPasteboardPromisedData adds a promise
  - Fulfilled in a promise keeper

```
#define kPasteboardPromisedData (CFDataRef)NULL

OSStatus PasteboardPutItemFlavor(
            PasteboardRef           inPasteboard,
            PasteboardItemID        inItem,
            CFStringRef             inFlavorType,
            CFDataRef               inData, //can be NULL
            PasteboardFlavorFlags inFlags );
```



Keeping Promises

- Install promise keeper before making promises
- Keep promise with PasteboardPutItemFlavorData
- Promises resolved when releasing PasteboardRef

```
OSStatus (*PasteboardPromiseKeeperProcPtr)(
            PasteboardRef     pasteboard,
            PasteboardItemID  item,
            CFStringRef       flavorType,
            void*             context );

OSStatus PasteboardSetPromiseKeeper(
            PasteboardRef                 inPasteboard,
            PasteboardPromiseKeeperProcPtr inKeeper,
            void*                         inContext );
```

Drag Additions

- Pasteboard support in Drag Manager
- Drag Flavor APIs unnecessary
- Get pasteboard from existing drag
- Create a new drag with existing pasteboard

```
OSStatus GetDragPasteboard(
            DragRef         inDrag,
            PasteboardRef*  outPasteboard );

OSStatus NewDragWithPasteboard(
            PasteboardRef   inPasteboard,
            DragRef*        outDrag );
```

Guidelines

- Pay attention to flavor flags and ordering
- UTI comparisons for best results
- WWDC Seed: NSPasteboardTypes, existence only temporary
- See <HIServices/Pasteboard.h> for details
- PasteboardPeeker sample
  - ADC Member Site, Download Software area, WWDC 2003 menu

Translation Services

Translation Today

- Limited typing mechanism
- Legacy API
- Extension based translation plug-ins

Translation Services Overview

- Successor to Translation Manager
- Source and destination types are UTIs
- Extensible via filter services
- Core Foundation based

Translation Reference

- Proxy to a resource that converts data formats
- Contains all information required to perform translation
  - Source type
  - Destination type
  - File or data translation
- Translation discovery and execution are separated

Translation Discovery

- Flags indicate support for files and data

```
typedef struct OpaqueTranslationRef* TranslationRef;

OSStatus TranslationCreate(
        CFStringRef             inSourceType,
        CFStringRef             inDestinationType,
        TranslationFlags        inTranslationFlags,
        TranslationRef*         outTranslation );
```

Translation Discovery (Cont.)

- Handles detailed discovery needs

```
OSStatus TranslationCreateWithSourceArray(
            CFArrayRef        inSourceTypes,
            TranslationFlags  inTranslationFlags,
            CFArrayRef*       outDestinationTypes,
            CFDictionaryRef*  outTranslations );
```

Translation Accessors

- Use accessors to decide best translation

```
OSStatus TranslationCopySourceType(
            TranslationRef    inTranslation,
            CFStringRef*      outSourceType );

OSStatus TranslationCopyDestinationType(
            TranslationRef    inTranslation,
            CFStringRef*      outDestinationType );

OSStatus TranslationGetTranslationFlags(
            TranslationRef    inTranslation,
            TranslationFlags* outTranslationFlags );
```

Translating Data in Memory

- Execute a translation created during discovery
  - TranslationRef must support data translations
  - Source and destination types must match TranslationRef
- Release destination data

```
OSStatus TranslationPerformForData(
            TranslationRef  inTranslation,
            CFDataRef       inSourceData,
            CFDataRef*      outDestinationData );
```

Translating Files

- TranslationRef must support file translations
- Location and name for the destination file settable

```
OSStatus TranslationPerformForFile(
  TranslationRef inTranslation,
  const FSRef*   inSourceFile,
  const FSRef*   inDestinationDirectory,//can be NULL
  CFStringRef    inDestinationName, //can be NULL
  FSRef*         outTranslatedFile );
```

Translating Files (Cont.)

- Only file URL translations supported
- Release URL reference

```
OSStatus TranslationPerformForURL(
        TranslationRef inTranslation,
        CFURLRef       inSourceURL,
        CFURLRef       inDestinationURL,//can be NULL
        CFURLRef*      outTranslatedURL );
```

Filter Services

- Default translations are extended by filter services
- Similar to Apple menu services with modified plist
- Performing translation
  - Service launched when necessary
  - Source data delivered via pasteboard
  - Translate
  - Put destination data back on pasteboard
- PasteboardPeeker sample
  - ADC Member Site, Download Software area, WWDC 2003 menu

Guidelines

- TranslationCreate for simple needs
- TranslationCreateWithSourceArray for complex needs
- See <HIServices/TranslationServices.h> for details

Summary

Get to Work!

- Uniform type identifiers provide rich type management
  - Use the UTI API for conformance tests
  - Declare new types in your Internet domain
- Pasteboard APIs streamline Scrap and Drag usage
  - Look at leveraging communication with Cocoa
  - Look for automatic translations
- Update to Translation Services
  - Use filter services to extend default translations

Wrap Up

Roadmap

409 Carbon: The HIToolbox	Presidio Wed., 9:00am
307 Using Apple Tools for Carbon Development	Russian Hill Thu., 9:00am
416 Carbon: Optimizing Performance	Presidio Thu., 2:00pm
423 How to Write a Modern Carbon Application	Nob Hill Fri., 2:00pm
425 Carbon: HIView in Depth	Nob Hill Fri., 3:30pm

Who to Contact

Xavier Legros
Mac OS X Evangelist, Favorite French guy
xavier@apple.com

http://developer.apple.com/wwdc2003/urls.html

Q&A

Xavier Legros

Mac OS X Evangelist, Favorite French guy

xavier@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference